OpenFOAM and HPC

- HPC
- Scaling (Amdahl’s Law and efficient workload sharing)
- Parallel I/O
- Other Workflows (Job Farming in Uncertainty Quantification)
- Parallel OF Post-Processing using ParaView

What is HPC?

What do YOU understand under HPC?

What is HPC?

When you grow larger than your local resources:

- Runtime
- Memory (RAM, HD)

What is the 1st Rule of HPC?

What is the 1st Rule of HPC?

Avoid it, if possible!!

Because it is hard work!
And you pay for it!
(HPC resources are expensive; and requires time)

General HPC Systematics

(Assuming that your engineering/physical problem is set.)

- Start small! (Iteration/Approach)
- Double check!! (Correctness)
- Parallelize!! (Speed-up)
- Reduce!!! (Resources)
- Automate!!!! (Workflows)

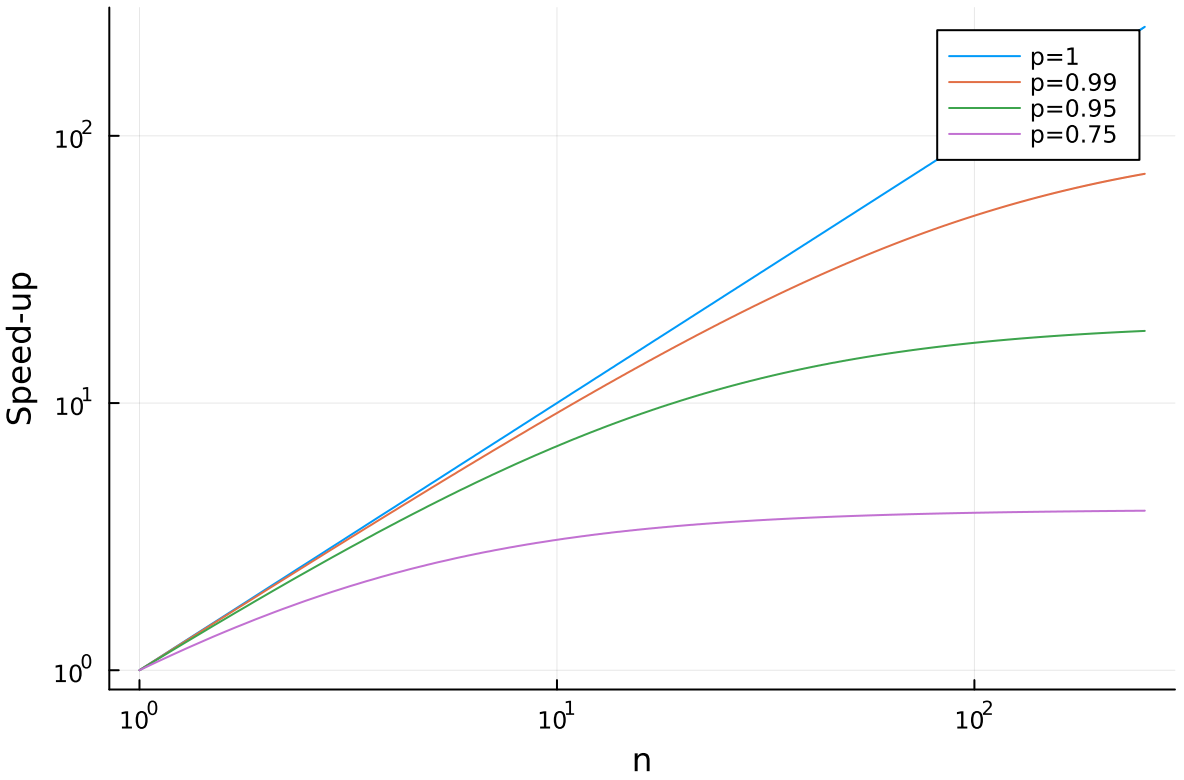
Strong Scaling - Amdahl's Law

Speed-up:

$$S_n = \frac{T_1}{T_n} = \frac{1}{(1-p)+p/n}$$

Parallel Efficiency:

$$\varepsilon_n = \frac{T_1/n}{T_n} = \frac{1}{n(1-p)+p}$$



Strong Scaling - OpenFOAM

- Work sharing and load balancing in OpenFOAM
  via **domain decomposition**
- metis, scotch, kahip, (hierarchical = built-in)
- *decomposeParDict* → **`numberOfSubdomains`**
- **`decomposePar`**, **`reconstructParMesh`** and **`reconstructPar`**
- OpenFOAM = **MPI only** (cfMesh MPI-OpenMP-hybrid)

Strong Scaling - OpenFOAM

Practical Approach:

$n =$	1	2	4	8	16	...
$T_n =$						
$T_1/n =$						
$\varepsilon_n =$						

“1” can be:
1 CPU
or 1 CPU/GPU
or 1 node
(or 10 nodes)

Amdahl’s Law - Drills

1. Take *incompressible/simpleFoam/motorBike* from **`$FOAM_TUTORIALS`** and perform scaling on the **`simpleFoam`** run!

   Remark: Check for not having <1000 cells/processor!

Amdahl's Law - Drills

Preparation – Case:

a) Copy use case to **`$SCRATCH`**

```
login> cp -r $FOAM_TUTORIALS/incompressible/\
simpleFoam/motorBike $SCRATCH/
```

b) Look into *Allrun* script, and prepare the case, until **`decomposePar`**!

c) Create a Slurm script (one can re-use the *Allrun* script) for the scaling analysis in the *motorBike* directory, and submit it!
Use 1, 2, 4, 8, 16 CPUs (MPI ranks)!

d) Read out the results, and plot the parallel efficiency!
Interpret these results!
(Extra: How long would a 2 Node run with each 28 CPUs take?)

Amdahl's Law - Drills

Preparation – Basic Slurm Script:

```
#!/bin/bash
#SBATCH -J job_name                 # job's name
#SBATCH -o ./%x.%j.%N.out           # output file stdout/err
#SBATCH -D ./                       # work dir == submit dir
#SBATCH --clusters=cm2_tiny         # which cluster?
#SBATCH --partition=cm2_tiny        # which partition?
#SBATCH --mail-type=none            # others possible
#SBATCH --export=NONE               # mandatory!!!!
#SBATCH --nodes=1                   # resource (1 node)
#SBATCH --ntasks-per-node=28        #
#SBATCH --time=00:10:00             # resource (10 minutes)
#SBATCH --reservation=hopf1w22      # reservation

module load slurm_setup             # mandatory @ LRZ

module use /lrz/sys/share/modules/extfiles/
module load openfoam/v2112-icc-impi_bashrc

mpiexec -n ? simpleFoam -parallel  # … <- your task!
```

Amdahl’s Law - Drills

Preparation – Hints:

- after **`snappyHexMesh`**, use **`reconstructParMesh`** to reconstruct the mesh

  or

- after **`checkMesh`**, use **`reconstrucPar`** (**`–withZero`**), in order to start for scaling from a reconstructed scenario
  (Don’t forget to remove *processor** directories between scaling runs!)

Amdahl’s Law - Drills

Preparation – Hints:

- **`redistributePar`** is supposed to be a **parallel** (!) and more efficient replacement for **`decomposePar`**, **`reconstructParMesh`**, and **`reconstructPar`**

Weak Scaling – Large Cases

- # MPI ranks AND # mesh cells are increased
  (both workload and resources are increased)

- Check mesh! Start with few time steps!
  (Does it scale as expected? Memory consumption
  about the same per node? Load-Balance (AMR)?)

Placement/Pinning matters …

- MPI Rank/OpenMP Thread → CPU !!!
  (Often seen problem: all ranks on one CPU if Slurm script mis-configured. Check!!)

- OpenMP Threads: memory locality important
  (within NUMA domain)

Job Control

- runtime control – controlDict:

  `runTimeModifiable true;`

- setting **`endTime`** < current time step (**`stopAt`**)
  → stop after next time step finished.
- **`writeControl timeStep; writeInterval 1;`**
  → writing after next time step.
- check **`processor*`** directories (parallel runs)

Advanced OpenFOAM I/O

a) ASCII/Binary – *controlDict*

```
writeFormat ascii/binary;
writeCompression on/off;    // ASCII
writePrecision … ;          // ASCII
```

b) Collated I/O

```
export FOAM_IORANKS="(0 4)"
decomposePar -fileHandler collated
mpiexec Solver -parallel -fileHandler collated
reconstructPar
```

Advanced OpenFOAM I/O

c) Checkpointing

controlDict: **`purgeWrite 2;`**

keeps last two written time steps

Compromise between writing too much/often and loosing data/CPU-h due to job failure/ I/O
(fast advancement ↔ saving valuable data)

HPC Workflow Issues

a) N single tasks; t_{task} small, n_{task} small, N large

b) N single tasks; t_{task} large, n_{task} small, N small
→ ergodic hypothese (short: time avg == ensemble avg) → a)

Pre/Post Processing Workflow

Demo:

- ParaView Server-Client (SSH/VNC)
  - ParaView Mesa **`pvserver`** on Compute Nodes
  - Memory Inspector
  - *cellDist* and *vtkProcessorId*
- **`pvpython`**/**`pvbatch`** (*Tools → Start/Stop Trace*)

Thank you for your Feedback!

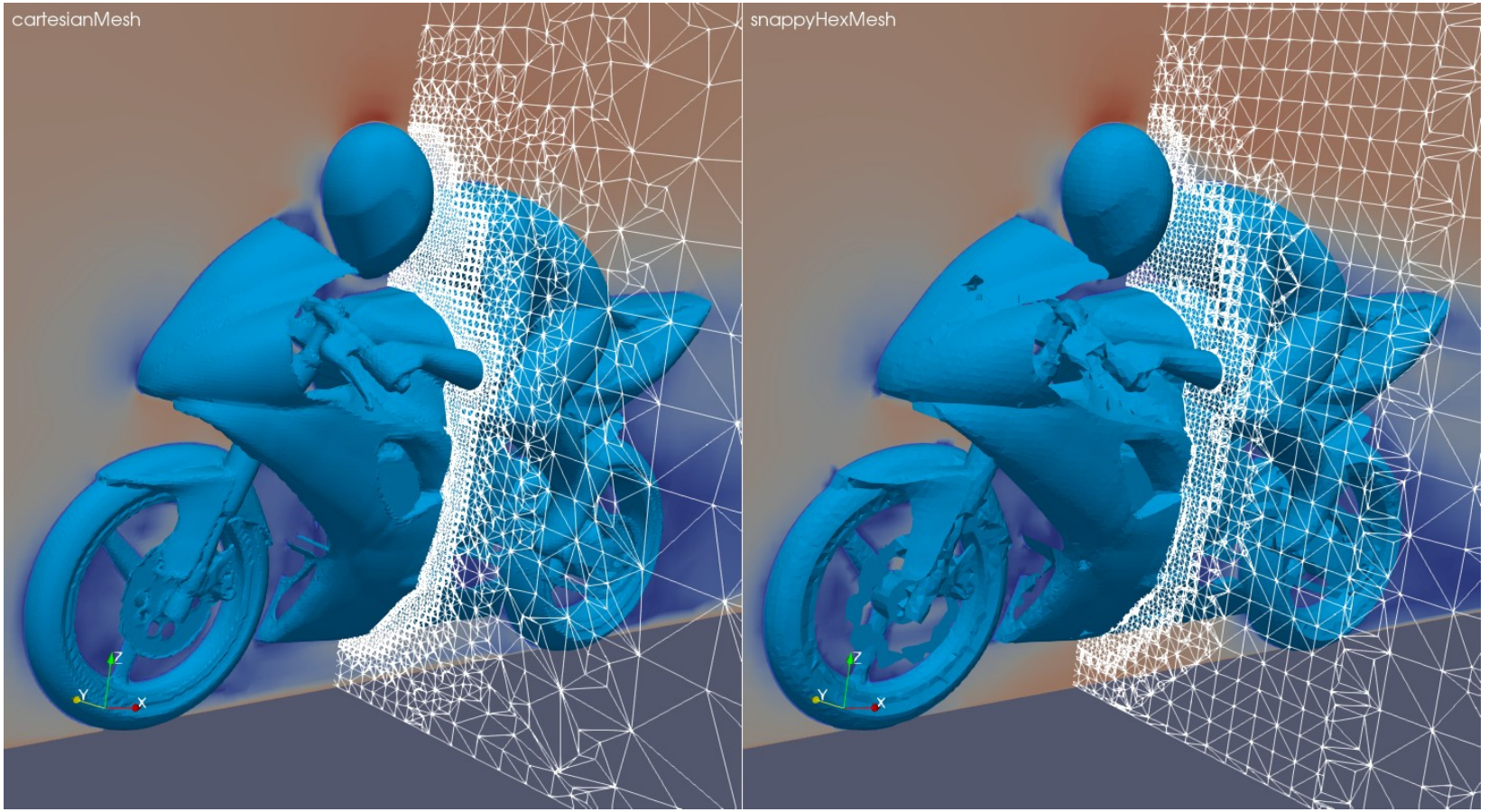
https://survey.lrz.de/index.php/456185?lang=en

cfMesh motorBike - Drill

- https://cfd-training.com/2020/04/29/how-to-use-cfmesh-a-first-tutorial-based-on-the-ahmed-body/
- https://www.youtube.com/watch?v=e2N-h0e_SmM
- https://cfmesh.com/wp-content/uploads/2015/09/User_Guide-cfMesh_v1.1.pdf

- Transform tutorial case incompressible/simpleFoam/motorBike to use **`cartesianMesh`** instead of **`snappyHexMesh`**!

cfMesh motorBike



cfMesh motorBike - Allrun

```
...
cp -f "$FOAM_TUTORIALS"/resources/geometry/motorBike.obj.gz .
gunzip motorBike.obj.gz

module load paraview-prebuild/5.10.0_mesa        # obj → stl
pvpython -c "from paraview.simple import *; motorBikeobj =
WavefrontOBJReader(registrationName='motorBike.obj',
FileName='motorBike.obj');SaveData('motorBike.stl', proxy=motorBikeobj,
CellDataArrays=['GroupIds'], FileType='Ascii')"
module rm paraview-prebuild/5.10.0_mesa

sed -i '1csolid motorBikeGroup' motorBike.stl
surfaceGenerateBoundingBox motorBike.stl motorBikeBB.stl 4.70834 14.2488 \
                                                         3.64971 3.66773 0 6.64848
cat motorBike.stl motorBikeBB.stl > geometry.stl

export OMP_NUM_THREADS=$(nproc --all)
runApplication cartesianMesh   # equivalent to "OMP_NUM_THREADS=X cartesianMesh"
...
```

cfMesh motorBike - meshDict

```
/*--------------------------------*- C++ -*----------------------------------*\|
| =========                 |                                                 |
| \\      /  F ield         | cfMesh: A library for mesh generation           |
|  \\    /   O peration     |                                                 |
|   \\  /    A nd           | Author: Franjo Juretic                          |
|    \\/     M anipulation  | E-mail: franjo.juretic@c-fields.com             |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version   2.0;
    format    ascii;
    class     dictionary;
    location  "system";
    object    meshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

surfaceFile "geometry.stl";
maxCellSize 1.25;

boundaryCellSize 1.25;

objectRefinements
{
    mainBox
    {
        type box;
        cellSize         0.25; //[m]
        centre       (3.5 0 0.75); //[m]
        lengthX          9.0; //[m]
        lengthY          1.0; //[m]
        lengthZ          2.0; //[m]
    }
}
```

```
localRefinement
{
    motorBike
    {
        additionalRefinementLevels    6;
        refinementThickness        0.25; //[m]
        cellSize                   0.05; //[m]
    }
}

surfaceMeshRefinement
{
    motorBike
    {
         surfaceFile "motorBike.stl";
         additionalRefinementLevels      7;
         cellSize                    0.015;//[m]
         refinementThickness         0.006;
    }
}


boundaryLayers
{
    patchBoundaryLayers
    {
        motorBike
        {
             nLayers                    10;
             thicknessRatio              1;
             allowDiscontinuity          1;
             maxFirstLayerThickness 0.001;
        }
    }
```

cfMesh motorBike - meshDict

```
    OptimiseLayer 1;

    OptimisationParameters
    {
        NSmoothNormals           5;
        RelThicknessTol        0.2;
        FeatureSizeFactor      0.4;
        ReCalculateNormals       1;
        MaxNumIterations         5;
    }
}

renameBoundary
{
    defaultType     wall;

    newPatchNames
    {
        "xMax" { newName outlet       ; type  patch; }
        "xMin" { newName inlet        ; type  patch; }
        "yMax" { newName frontAndBack ; type  patch; }
        "yMin" { newName frontAndBack ; type  patch; }
        "zMax" { newName upperWall    ; type  patch; }
        "zMin" { newName lowerWall    ; type  wall; }
    }
}

// ************************************************************************* //
```

patch labels and types must match to boundary conditions (*0.orig/**)

Thank you for your Feedback!

https://survey.lrz.de/index.php/456185?lang=en